
PROJET

Puissance 4

Le but de ce projet est de réaliser un programme permettant à l'utilisateur de jouer au Puissance 4 contre l'ordinateur.

1 Travail à Rendre

Le travail peut être effectué en binôme. Vous devez rendre le code source de votre programme. Pour chacune des fonctions, vous indiquerez dans l'entête une description rapide de l'action de la fonction, le type du résultat qu'elle retourne ainsi que le type et le rôle des arguments. En plus de cela vous joindrez un rapide mode d'emploi présentant votre programme et précisant comment il fonctionne.

Une bonne décomposition de la programmation sera prise en compte.

2 Introduction

(Tiré de http://fr.wikipedia.org/wiki/Puissance_4)

Puissance 4 est un jeu de stratégie combinatoire abstrait, commercialisé pour la première fois en 1974 par la Milton Bradley Company, plus connue sous le nom de MB et détenue depuis 1984 par la société Hasbro.

Le but du jeu est d'aligner 4 pions sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur dispose de 21 pions d'une couleur (par convention, en général jaune ou rouge). Tour à tour les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans ladite colonne et c'est ensuite à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) d'au moins quatre pions de sa couleur. Si alors que toutes les cases de la grille de jeu sont remplies aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.

Depuis 1988, il est établi, suite à l'analyse informatique exhaustive du jeu, que le joueur qui commence la partie gagnera toujours s'il joue les coups adéquats. De nos jours, un certain nombre de programmes informatiques sont capables de jouer parfaitement à Puissance 4 et donc de gagner systématiquement dès lors qu'ils entament la partie. Les plus connus sont Mustrum, 4 in a row (lorsqu'il est réglé au niveau infini), Four or more, Velen, Vianiato, ConnectFour3D, TitOT ou encore Conny. Hormis Vianiato et Conny, ils sont tous téléchargeables gratuitement à partir d'internet.

Le jeu peut sembler simpliste au premier abord, car il est possible de prévoir, sans l'aide d'un ordinateur, un grand nombre de coups pour ainsi gagner la partie. Cependant, il faut compter avec les décisions de l'adversaire.

Le principe de base est de placer les jetons de préférence dans la colonne centrale. Tout jeton dans celle-ci peut former un grand nombre de lignes dans diverses directions en longueur, et aussi retirer du même coup cette possibilité à l'adversaire.

Je vous conseille également pour bien vous imprégner du principe du jeu, si vous n'y avez jamais joué de faire quelques parties en lignes à l'adresse suivante : <http://jeux.prise2tete.fr/puissance-4/puissance-4.php>

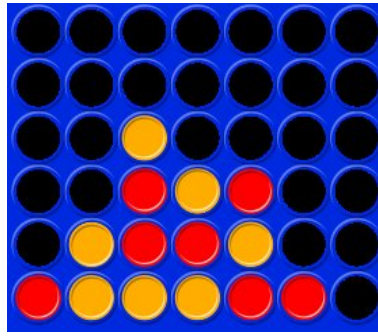


FIGURE 1 –

3 Puissance 4 et *Racket*

Dans le projet que nous allons réaliser, nous aurons à gérer une structure de données correspondant au plateau de jeu ainsi que les mouvements corrects des joueurs (humain et ordinateur).

Dans la première version de ce petit projet, nous nous occuperons également de programmer le comportement de l'ordinateur qui sera très simple : il choisira ses coups de manière aléatoire. Vous pourrez aisément modifier ce comportement pour l'améliorer.

3.1 Structure de données pour la plateau de jeu

Je vous propose la structure de données suivante pour gérer le plateau de jeu. Vous n'êtes bien évidemment absolument pas obligé d'utiliser cette structure et vous pouvez définir la vôtre.

Le premier joueur jouera avec des X, le second joueur jouera avec des O.

Le plateau est représenté par une liste comprenant **deux** éléments. Le premier élément de la liste est une **chaîne de caractères** qui contient la totalité du plateau de jeu sous forme textuelle. Cette chaîne de caractères contient successivement la rangée supérieure (rangée 6), puis la rangée 5 et ainsi de suite jusqu'à la rangée 1. Les rangées sont séparées par un |, il y a donc exactement 47 caractères dans cette chaîne. Les colonnes sont numérotées de 1 à 7 de gauche à droite.

Le choix d'utiliser des caractères permet d'utiliser de nombreuses fonctions *Racket* manipulant les chaînes de caractères. Les espaces indiqueront l'absence de jeton, un X ou un O un des jetons d'un des joueurs. Cette partie de la structure sera appelée **chaîne-plateau**.

A l'état initial cette chaîne a la valeur suivante :

```
"      |      |      |      |      |      "
```

Si le joueur X pose un jeton dans la colonne 2, le résultat sera le suivant :

```
"      |      |      |      |      | X     "
```

Note : un caractère en *Racket* est noté de la manière suivant `#\c` pour le caractère `c`.

La deuxième partie de la liste gérant le plateau est une simple liste de 7 entiers correspondant au nombre de jetons dans les colonnes 1 à 7 (la colonne 1 étant à gauche). Cette deuxième partie de la structure sera appelée **liste-pieces**.

En résumé, le plateau a la structure suivante :

```
plateau = (chaîne-plateau liste-pieces)
```

A l'état initial, le plateau est défini par la variable globale suivante :

```
(define p4-start
  (list "      |      |      |      |      |      " '(0 0 0 0 0 0 0)))
```

Créer des plateaux différents et donner la structure du plateau correspondante.

4 Travail à réaliser

Donnez la représentation de la variable plateau pour le jeu correspondant à la figure 1.

4.1 Fonctions outils

Nous aurons besoin d'un certain nombre de fonctions outils pour réaliser ce projet. En voici quelques unes à réaliser, vous en aurez sans doute d'autres à écrire.

1. écrire une fonction **chaîne-plateau** qui retourne une partie d'un plateau, la partie chaîne de caractères du plateau ;

```
> (chaîne-plateau p4-start)
"      |      |      |      |      |      "
```

2. de même écrire une fonction **liste-pieces** retournant la seconde partie d'un plateau ;

```
> (liste-pieces p4-start)
(0 0 0 0 0 0 0)
```

3. écrire une fonction **liste-ref** permettant d'accéder au i^eélément d'une liste passée en paramètre.

4.2 Gestion du plateau

Nous allons avoir besoin d'accéder aux différents éléments du plateau. Dans cette partie, vous aurez besoin de la fonction prédéfinie **string-ref** retournant le i^ecaractère d'une chaîne. Attention, en *Racket*, les chaînes commencent à l'indice 0.

```
> (string-ref "abcde" 2)
#\c
```

1. La première fonction dont nous aurons besoin est une fonction retournant la position dans la chaîne de caractères de la variable **plateau** d'un élément connaissant sa ligne et sa colonne. Écrire une fonction **position-piece-chaîne** retournant la position dans la chaîne de caractères d'un élément connaissant sa ligne et sa colonne.

```
> (position-piece-chaîne 1 1)
40
> (position-piece-chaîne 6 1)
0
```

Ceci indique que le jeton à la 6^eligne 1^{re}colonne est à la position 0 dans la chaîne de caractères du plateau.

2. écrire une fonction **plateau-caractere** (fonction à 3 paramètres) retournant le caractère situé à la ligne et à la colonne passées en paramètre.

```
> (plateau-caractere p4-start 2 3)
#\space
```

Cette fonction retourne donc soit le caractère espace, soit "X", soit "O".

3. en utilisant la fonction précédente écrire une fonction **piece-a** (fonction à 3 paramètres) qui retourne la pièce située aux rang et colonne passés en paramètre. Cette fonction retourne soit **X**, soit **O** ou **vide**

```
> (piece-a p4-start 2 3)
vide
```

4. la fonction suivante est le cœur de la gestion du plateau. Elle retourne le plateau modifié après l'ajout d'un jeton à une colonne donnée.

Vous aurez également besoin des fonctions intermédiaires suivantes :

substring qui retourne une portion d'une chaîne de caractères entre 2 indices

```
> (substring "abcde" 0 2)
"ab"
> (substring "abcde" 1 3)
"bc"
```

string-length qui retourne la taille globale d'une chaîne :

```
> (string-length "abcde")
5
```

et **string-append** qui concatène plusieurs chaînes de caractères :

```
> (string-append "il " "fait " "beau.")
"il fait beau."
```

En utilisant toutes ces fonctions et en définissant éventuellement d'autres fonctions si nécessaires, écrire la fonction **joue-jeton** prenant en paramètre un plateau, un joueur et une colonne qui retourne le plateau après modification :

```
> (joue-jeton p4-start 'X 2)
("      |      |      |      |      | X      " (0 1 0 0 0 0 0))
> (joue-jeton p4-start 'O 5)
("      |      |      |      |      |      O      " (0 0 0 0 1 0 0))
```

On supposera pour le moment que les paramètres passés à cette fonction sont correctes.

Vous aurez certainement besoin d'écrire un certain nombre de fonctions annexes pour écrire la fonction **joue-jeton**. Je vous conseille d'écrire les deux fonctions outils suivantes qui vous seront utiles :

modif qui permet de modifier le ième élément d'une liste

```
> (modif '(1 2 3 4 5) 2 8)
'(1 8 3 4 5)
```

et la fonction **symbol->chaîne** qui permet de transformer respectivement le symbole X ou O en la chaîne "X" ou "O".

```
> (symbol->chaîne 'X)
"X"
> (symbol->chaîne 'O)
"O"
```

4.3 Validité d'un coup

Écrire une fonction `mouvement-valide` qui pour un plateau passé en paramètre retourne l'ensemble des mouvements valides (c'est-à-dire l'ensemble des colonnes où on peut déposer un jeton).

```
> (mouvement-valide p4-start)
(1 2 3 4 5 6 7)
```

Ceci indique que l'on peut déposer un jeton dans les colonnes 1 à 7.

4.4 Calcul des coups potentiels

Écrire une fonction `p4-enfants` qui retourne une liste de tous les plateaux *enfants* qui peuvent résulter d'un mouvement valide. (Vous avez tout intérêt à utiliser la fonction `map` pour cette fonction).

Cette liste de coups nous permettra de décider de la stratégie de l'ordinateur.

```
> (p4-enfants p4-start 'X)
(("      |      |      |      |      |X      " (1 0 0 0 0 0 0))
 ("      |      |      |      |      | X      " (0 1 0 0 0 0 0))
 ("      |      |      |      |      |  X      " (0 0 1 0 0 0 0))
 ("      |      |      |      |      |   X      " (0 0 0 1 0 0 0))
 ("      |      |      |      |      |    X      " (0 0 0 0 1 0 0))
 ("      |      |      |      |      |     X      " (0 0 0 0 0 1 0))
 ("      |      |      |      |      |      X      " (0 0 0 0 0 0 1)))
```

Ceci indique qu'à l'état initial, le joueur **X** peut jouer dans toutes les colonnes (1 à 7).

4.5 Tests de fin de partie

La fonction suivante `p4-end?` teste si la fin de partie est atteinte pour un plateau donné. Étudier la et expliquer sur votre compte-rendu son principe de fonctionnement.

```
1 ;; teste si la fin de partie est atteinte en utilisant
  ;; les expressions regulieres
  (define p4-end?
    (lambda (plateau)
      (let ((p (chaine-plateau plateau)))
6        (cond
          ((or (regexp-match (regexp "XXXX") p)
               (regexp-match (regexp "X.....X.....X.....X") p)
               (regexp-match (regexp "X.....X.....X.....X") p)
               (regexp-match (regexp "X.....X.....X.....X") p))
11          'X)
          ((or (regexp-match (regexp "OOOO") p)
               (regexp-match (regexp "O.....O.....O.....O") p)
               (regexp-match (regexp "O.....O.....O.....O") p)
               (regexp-match (regexp "O.....O.....O.....O") p))
16          'O)
          ((and (= 6 (first (liste-pieces plateau)))
                 (apply = (liste-pieces plateau))))
          'nul)
```

```

21         (else #f)
        )
    )
)

```

4.6 Joueur humain et ordinateur

Programmer une fonction `joueur-humain` qui demande au joueur une colonne, qui vérifie que la colonne rentrée est valide et qui appelle la fonction `joue-jeton`.

De même, écrire une fonction `joueur-ordi` qui jouera l'ordinateur à partir des listes de coups potentiels rendues par la fonction `p4-enfants`. Pour une première tentative, vous pouvez faire jouer l'ordinateur au hasard mais il est très simple d'améliorer cette fonction.